Eclipse Plug-in Development Research Project

Report 8

UNIVERSITY AT BUFFALO

August 17, 2015

Authored by: Chern Yee Chua

Eclipse Plug-in Development Research Project

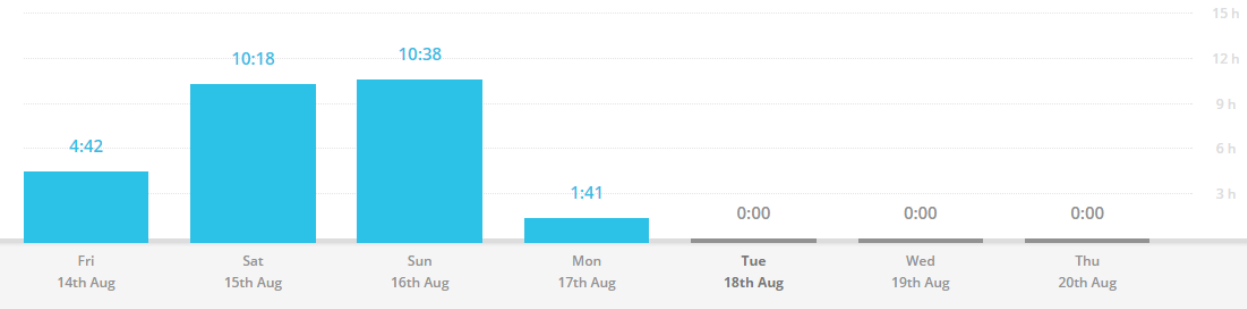
Report 8

Introduction

I have come up with two methods in dealing with the changes of AST node. The details of the methods will be discussed in the section below.

Synopsis

I spend more than 25 hours on this project for this meeting.



Eclipse Plugin	27:19:00
Analyzing the structure of AST	4:42:00
Constructing method that returns all the the child AST node and define OurASTMatcher	4:41:00
Report writing and some tiny fixing	1:41:00
Reworking on the node with line numbers	6:32:00
Visiting each specific node and comparing the nodes	4:06:00
Working on printing out each node with line numbers	5:37:00

Explanation of content

So far, I have come up with *two* methods to handle the changes of AST node.

1. Visit the specific node and check if the subtree is identical. If the subtree is not identical, print out that node and also its related child node.
2. Get the two AST root nodes from source, store each child node into array list of string based on the root node. Compare these two array lists and return the differences. In this method, I also capture the line number of the node. I also have setup the minimum wait time for each changes made in the editor.

Implementation: Method 1

My idea for **Method 1** is to do a subtreeMatch on those two AST root nodes using the user-defined AST Matcher to return the part of the node that returns false. Below is my implementation for **Method 1**:

a. Create a user-defined ASTMatcher, named OurASTMatcher that overrides **all** the match() methods from the ASTMatcher class.
b. Refactoring those methods so that when a false is returned, that means the subtree is not matched and then print out that specific node.

I did trace back the node that is printed out and realize that there are types of node that I wish to be printed out but it didn't. For example, a simple declaration of [int a;] should be fallen under

TypeDeclaration > BodyDeclaration > FieldDeclaration > PrimitiveType

for the [int] and

TypeDeclaration > BodyDeclaration > FieldDeclaration > VariableDeclarationFragment > SimpleName

for the [a] according to the ASTView.

But when I do the subtreeMatch of two AST root nodes, the **FieldDeclaration** is never being visited for some reasons. When I do the testing, I found out that it is actually fallen under TypeDeclaration and CompilationUnit. As a result, when I return the node of TypeDeclaration or CompilationUnit, it returns everything inside the editor which is something that we don't want.

So, I use **FieldDeclaration** as example and use ASTVisitor to just purposely visit the FieldDeclaration node to see what is going on. It turns out that the FieldDeclaration node is visited and it displays all the FieldDeclaration nodes.

Realizing that the nodes are incomparable using the default built-in method even though the structure of the nodes are exactly the same, I decided to use subtreeMatch on those specific nodes, going over those nodes one by one. Finally, I am able to extract the FieldDeclaration nodes that are not in the other list.

The modified procedure goes like this:

a. Let the two AST root nodes accept the ASTVisitor.
b. For whatever node that is visited, store those nodes into ArrayList according to the type of AST nodes.
c. Compare those specific type ArrayList, see which list is larger. If the former list is larger, that means there are elements being removed. Otherwise, there are elements added in.
d. Indicate whether element is ADDED or REMOVED and then swap the lists if the former list is larger.
e. Use two for-loops to remove redundant items of the larger list.
f. Use for-loop to print out all the nodes that is left in the list.
g. Clear those lists so that elements won't stack up.

The advantage of **Method 1** is that I can retrieve any information from the specific node that I obtained which gives us much more flexibility on the information that we want to display.

The drawback is that the amount of work will be significantly larger as I will need to implement all those unimplemented methods individually and the implementation varies differently.

As for this research project's purposes, we don't care too much about the cost and running time, so it will be a better solution if it is implemented fully and correctly.

If professor prefers this way, then I will need to implement those unimplemented methods, not all but just those that matter to this project.

Implementation: Method 2

Method 2 is rather an easier approach than **Method 1**. Basically, I just need to obtain the two consecutive AST root nodes and pass them into a print() function that generally visits every single child node and store the information of each node into string array list respectively. Comparing these two array list and then outputs the differences.

Known issues with this implementation:

- Whenever the user add or remove item that is causing an addition or removal of lines, the subsequent item beneath the changes will be affected. For example,

```
1  package issacpackage;
2
3  public class IssacClass {
4      int a;
5      int b;
6      int c;
7
8
9      public void method(){
10         int a;
11     }
12
13
14
15 }
```

If I remove the line int b (line 5), the int c will be at line 5, and the method() will be at line 8 and so on and my code will think that is a new added elements since the line number is changed.

Of course, I can fix this issue by comparing the corresponding strings. For example, if the discrepancy of number character in the compared string is less than 3 (or 5 depending the number of total lines), then the string is said to be the same and it won't be shown in ADDED or REMOVED.

Conclusion

I will continue working on whichever method that professor likes. If all that we need is just the line number together with that specific node, I would say Method 2 is better. But if we want to retrieve more than just the basic information of the node, I would think Method 1 is better.

Thank you!